PROCÉDÉ D'ORGANISATION D'UNE BASE DE DONNÉES

La présente invention se rapporte au domaine des bases de données. La présente invention porte plus particulièrement sur un procédé technique d'organisation d'une base de données.

On connaît dans l'art antérieur, par la demande de brevet américain US 2004/0098363 (IBM), un système de stockage hiérarchique de données. Des objets de données sont stockés dans une hiérarchie de stockage et des tables de contenu contenant des entrées sont générées. L'emplacement des tables de contenu est géré de façon dynamique.

L'art antérieur connaît également, par la demande de brevet européen EP 1 423 799 (Lafayette Software) des procédés pour organiser des données et réaliser des requêtes dans un système de bases de données. Les informations sont organisées dans un système de bases de données avec des groupes d'attributs définis et des mots de collection de données assignés aux attributs en associant une liste d'identifiants de graphes de données avec une entrée de thésaurus.

L'art antérieur connaît également par la demande de brevet PCT WO 04/25507 (Karmic Software Research), qui correspond à la demande de brevet français FR 2 844 372, un procédé d'organisation d'une base de données numériques sous une forme traçable. Plus précisément, cette demande porte sur un procédé d'organisation d'une base de données numériques sous une forme traçable, comportant des étapes de modification d'une base de données numériques principale par ajout ou suppression ou modification d'un enregistrement de la base principale et des étapes de lecture de la base de données principale, caractérisé en ce que

- l'étape de modification de la base de données principale comprend une opération de création d'au moins un enregistrement numérique comportant au moins :
  * les identifiants numériques uniques des enregistrements et des attributs concernés de la base de données principale,
  * un identifiant numérique unique de l'état de la base de données principale correspondant à ladite modification de la base de données principale,
  * les valeurs élémentaires des attributs qui leur sont affectées à travers les opérations élémentaires, sans procéder au stockage des attributs ou des enregistrements non modifiés,
- et d'ajout dudit enregistrement dans une base d'historisation interne composée d'au moins une table,
- et en ce que l'étape de lecture portant sur tout état final ou antérieur de la base de données principale consiste à recevoir (ou intercepter) une requête originelle associée à l'identificateur unique de l'état visé, à procéder à une transformation de ladite requête originelle pour construire une requête modifiée d'adressage de la base d'historisation comprenant les critères de la requête originelle et l'identificateur de l'état visé, et de reconstruction du ou des enregistrements correspondant aux critères de la requête originelle et à l'état visé, ladite étape de reconstitution consistant à retrouver les valeurs élémentaires, contenues dans les enregistrements de la base d'historisation, correspondant aux critères de la requête originelle [afin de réduire les besoins de capacité de stockage et les temps de traitement].

On connaît également, par le brevet américain US 6 292 795 (IBM), un système d'indexation de fichiers et un mécanisme pour accéder à des données dans un tel système.

Enfin, on connaît aussi dans l'art antérieur le brevet américain US 5 826 262 (IBM) un procédé de construction en parallèle d'arbres à radicaux.

Le problème technique que la présente invention se propose de résoudre est celui qui consiste en l'amélioration des performances des requêtes dans une base de données. En effet, les procédés de l'art antérieur consomment de grandes ressources machine, que ce soit du point de vue des ressources processeurs que des ressources disques.

À cet effet, la présente invention concerne, dans son acception la plus générale, un procédé d'organisation d'une base de données relationnelle destiné à être mis en œuvre sur une architecture informatique comprenant au moins un processeur et de la mémoire, caractérisé en ce qu'il comporte les étapes consistant à :

- élaborer une table d'expansion hiérarchique ;
- créer un thésaurus sur chacune des colonnes ;
- pour chacun des mots de chacun des thésaurus, créer l'arbre des radicaux de l'ensemble des indices de lignes sur lesquelles ledit mot apparaît ;
- pour chacune des clés primaires, stocker la suite de ces valeurs en mettant en œuvre une permutation sur l'ensemble de ces valeurs dans le but de retrouver une donnée.

Avantageusement, le procédé comporte en outre une étape de découpage des tables de la base de données en un ensemble de sous-tables, chacune comportant un nombre donné de lignes à l'exception de la dernière sous-table.

De préférence, la base de données met en œuvre le langage SQL (Structured Query Language).

La présente invention se rapporte également à un système de base de données organisé selon le procédé d'organisation défini plus haut.

La présente invention se rapporte également à un procédé de requêtage d'une base de données organisée selon le procédé d'organisation défini plus haut, caractérisé en ce qu'il comporte

- une première étape de calcul d'une table d'expansion ;
- résoudre la clause « Where » de la requête en interrogeant les colonnes de ladite table d'expansion ;
- interroger les images non inversées des colonnes pour résoudre la clause « Select ».

On comprendra mieux l'invention à l'aide de la description, faite ci-après à titre purement explicatif, d'un mode de réalisation de l'invention, en référence aux figures annexées :

- la figure 1 illustre un stockage au moyen d'un arbre à radicaux ;
- la figure 2 illustre un exemple de représentation d'une colonne d'une table ;
- la figure 3 illustre un résumé du stockage complet d'une colonne ;
- les figures 4 et 5 illustrent un arbre à radicaux avant et après une opération NOT.

Un arbre à radicaux est un moyen pratique de stocker des ensembles d'entiers, particulièrement quand ils sont écrits sur une même longueur. Lorsqu'on utilise des entiers, il est clairement toujours possible de leur imposer la même longueur d'écriture (celle du plus long ou plus) en faisant précéder leur écriture du nombre adéquat de chiffres 0.

Considérons par exemple un ensemble d'entiers qu'on écrit sur une longueur unique en base 2, S = {0, 2, 5, 7, 11} = {0000, 0010, 0101, 0111, 1011}. On peut alors stocker cet ensemble dans un arbre à radicaux dont les chemins, depuis la racine jusqu'aux feuilles, représente l'écriture de l'entier stocké dans la feuille de l'arbre. Par exemple, l'ensemble précédent peut être stocké dans l'arbre à radicaux de la Figure 1.

Les avantages à utiliser un arbre à radicaux sont fort nombreux : le stockage est économique en termes d'espace mémoire car les préfixes communs aux nombres distincts ne sont stockés qu'une seule fois. Par ailleurs, comme nous le verrons dans les paragraphes suivants, les opérations logiques sur des ensembles stockés ainsi sont rapides, économiques en termes de ressources machines et simples à implémenter.

On détaille comment des arbres à radicaux peuvent être utiles et efficaces pour stocker les données d'une base de données ou pour la modifier.

Dans la première partie, on suppose que la base de données n'est composée que d'une table, elle-même composée d'une unique colonne.

Puis nous supposerons que la base de données n'est composée que d'une unique table, elle-même composée de plusieurs colonnes, et d'au moins une clé primaire. Il peut en effet s'avérer très pratique d'autoriser une table à être munie de plusieurs clés primaires. En effet, en pratique, il arrive fréquemment qu'une ligne de table ne soit que partiellement remplie. Il se peut alors qu'une des clés primaires soit incomplète, donc inutilisable, mais qu'une autre soit complète.

La dernière sous-partie est dédiée à la création des index d'une base de données quelconque.

Une clé primaire est une colonne, ou un ensemble ordonné de colonnes, tel que deux lignes différentes de la table ne puissent prendre les mêmes valeurs sur cette (ou ces) colonnes.

Il existe cependant toujours une clé primaire implicite et très utile : l'indice de la ligne dans la table (il s'agit en effet d'une clé primaire puisque deux lignes distinctes ne peuvent avoir le même indice de ligne). Dans la suite de la description de cette invention, nous supposerons que cette clé primaire est effective.

Si on doit stocker, interroger et gérer une base de données faite d'une unique table elle-même constituée d'une unique table, on peut calculer le thesaurus de cette colonne et pour chaque mot de ce thesaurus calculer l'ensemble des indices de lignes auxquels il apparaît.

Ces indices de lignes peuvent tout naturellement être stockés dans un arbre à radicaux.

Remarquons qu'au cours de la création du thesaurus, un tri des données est effectué. On trie en effet l'ensemble des couples (mot, indice de ligne) selon les mots et, à mot égal, selon les indices de lignes. Ainsi on peut d'une part calculer le thesaurus et d'autre part, pour chacun des mots de ce thesaurus l'arbre à radicaux des indices de lignes auxquels il apparaît.

Prenons un exemple : la table :

0	Male
1	Female
2	Female
3	Male
4	Female
5	Male
6	Male
7	Female
8	Female
9	Male
10	Male

(dans cet exemple, les indices de lignes sont indiqués explicitement).

On construit alors les couples

(Male, 0), (Female, 1), (Female, 2), (Male, 3), (Female, 4), (Male, 5), (Male, 6), (Female, 7), (Female, 8), (Male, 9), (Male, 10)

et on les trie selon leur premier élément en priorité :

(Female, 1), (Female, 2), (Female, 4), (Female, 7), (Female, 8),

(Male, 0), (Male, 3), (Male, 5), (Male, 6), (Male, 9), (Male, 10).

On peut alors construire le thesaurus et, pour chaque mot du thesaurus, l'ensemble des indices de lignes auxquels il apparaît.

Le mot ``Female'' apparaît aux lignes {1, 2, 4, 7, 8} et ``Male'' apparaît aux indices {0, 3, 5, 6, 9, 10}.

Après ce travail, il est très simple de répondre à une question comme ``Quels sont les indices de lignes auxquels apparaît le mot ``Male'' ?'' mais relativement difficile de répondre à une question comme ``Quel est le contenu de la cellule de la ligne 5 ?'' . Pour ce genre de requête, on pourra consulter la sous-section 5 ci-après.

Ces ensembles d'indices de lignes peuvent donc être stockés dans des arbres à radicaux. Ce procédé de stockage est très utile pour calculer l'intersection, la réunion etc. \dots de tels ensembles.

Dans l'exemple précédent, on obtient le résultat présenté Figure 2.

Il est une autre requête courante qui concerne le contenu d'une colonne : le ``entre'' : on peut souhaiter connaître les indices de lignes dont le contenu est compris entre deux bornes données.

Imaginons par exemple qu'une colonne contienne des dates, écrites au format AAAAMMJJ. Comparer deux dates stockées sous ce format est en fait la même chose que les comparer lexicographiquement.

Mais nous pouvons aussi enrichir le thesaurus de mots obtenus comme troncatures des mots du thesaurus initial. Par exemple, on peut décider d'enrichir le thesaurus de toutes les troncatures des quatre ou six premières lettres des mots du thesaurus initial.

Ainsi chaque mot serait représenté, dans notre exemple, trois fois : une fois en tant que lui-même, une

fois tronqué à six caractères et une dernière fois tronqué à quatre caractères.

Tout mot de six caractères, disons aaaamm, apparaîtra à chaque fois que la ligne initiale contenait un mot aaaammxx. En d'autres termes, l'ensemble des lignes auxquelles apparaîtra le mot aaaamm est la réunion des ensembles d'indices de lignes où apparaît un mot de la forme aaaammxx (c'est-à-dire aaaamm suivi de quoi que ce soit).

De même un mot de quatre caractères aaaa apparaîtra à chaque fois qu'un mot de la forme aaaaxxyy était présent dans la table initiale. Son arbre à radicaux est donc l'union des arbres à radicaux des mots dont il est préfixe.

L'intérêt est qu'une clause ``entre'' (*between* en anglais) peut se traiter avec une économie importante en termes de lectures sur procédé de stockage. Par exemple, si on cherche l'ensemble des lignes auxquelles apparaît une date comprise dans l'intervalle [19931117, 19950225], le nombre de lectures nécessaires d'arbres à radicaux est de 14+1+1+1+25 = 42 (car [19931117, 19950225] = [19931117, 19931130] U [199312, 199312] U [1994, 1994] U [199501, 199501] U [10050201, 19950225]), au lieu de 466.

Il peut parfois se trouver que certaines lignes d'une table ne soient pas renseignées. Mais pour créer les arbres à radicaux, toute ligne devrait avoir une valeur.

On choisit donc des valeurs signifiant que la ligne correspondante ne contient pas d'information. Naturellement, on choisira une valeur ayant un rapport avec le type des données stockées; \`A titre d'exemple, on peut choisir :

#Empty# pour une chaîne de caractères, -2^{31} pour un entier signé sur 32 bits, $2^{32}-1$ pour un entier non signé sur32 bits,

-2^{63} pour un entier signé sur 64 bits,

$2^{64}-1$ pour un entier non signé sur 64 bits, etc.

Comme expliqué précédemment, le stockage d'une colonne par thesaurus et arbres à radicaux n'est pas très pratique pour répondre à une requête comme ``Quelle est la valeur de la ligne 17 ?''. par exemple.

C'est pourquoi il est nécessaire de stocker en sus la colonne dans son ordre naturel. Bien sûr, plutôt que stocker la colonne elle-même, il sera souvent avantageux de stocker la suite des indices de mots dans le thesaurus. Nous nommons ce stockage supplémentaire l'image non inversée de la colonne.

Par exemple, la colonne précédente sera stockée de la façon suivante :

Thesaurus

0	Female
1	Male

et la colonne :

0
1
1
0
1
0
0
1
1
0
0

Remarque : il peut se trouver qu'au fur et à mesure que la base de données est transformée, un mot apparaisse ou disparaisse du thesaurus (par exemple lorsqu'on retire ou ajoute des lignes à la table). On pourrait dès lors penser que la réécriture complète de la colonne est nécessaire. Ce n'est en fait pas le cas : Plutôt que stocker le thesaurus trié, on peut le stocker non trié et enregistrer à part une permutation permettant de retrouver l'ordre lexicographique des mots qui le composent. C'est pourquoi si un mot apparaît dans le thesaurus, la réécriture complète de la colonne n'est pas nécessaire. On réécrit dans ce cas la permutation permettant de retrouver l'ordre lexicographique des mots plutôt que le thesaurus lui-même.

La figure 3 illustre le résumé du stockage complet d'une colonne.

Dans le cas d'une base de données ne contenant qu'une seule table elle-même constituée de plusieurs Colonnes peut être traitée comme si elle était formée de colonnes indépendantes. En d'autres termes, on peut créer le stockage de chacune des colonnes constituant la table.

La seule question en suspens est alors le traitement de la clé primaire.

Lorsqu'on s'occupe d'une clé primaire, on a besoin d'être capable de répondre le plus rapidement possible à deux types de requêtes opposées : ``À quelle ligne peut-on trouver une valeur donnée de clé primaire'' et ``Quelle est la valeur de la clé primaire trouvée à une ligne donnée ?''.

On peut répondre efficacement à ces deux types d'interrogation en stockant à la fois la colonne ou les colonnes formant la clé primaire dans l'ordre dans lequel

les lignes apparaissent dans la table et une permutation permettant de lire les colonnes dans l'ordre lié à une fonction quelconque de comparaison. On peut alors retrouver une valeur donnée par dichotomie.

Par exemple, imaginons qu'une clé primaire est formée de deux colonnes, dont les valeurs sont stockées dans le tableau ci-dessous.

(0)	1	3
(1)	2	1
(2)	3	2
(3)	2	3
(4)	1	2
(5)	3	7
(6)	2	2
(7)	1	1
(8)	3	3
(9)	4	3

Dans cet exemple, les indices de lignes sont à nouveau exprimés explicitement mais mis entre parenthèses. On stocke donc les deux colonnes exactement comme elles sont dans la table et une permutation, liée à une fonction de comparaison qu'on choisit. Par exemple, on peut décider de comparer d'abord la première colonne lexicographiquement et à valeur égale comparer la deuxième ordinalement.

Dans ce cas, la clé primaire triée est :

(7)	1	1
(4)	1	2
(0)	1	3
(1)	2	1
(6)	2	2
(3)	2	3
(2)	3	2
(8)	3	3
(5)	3	7
(9)	4	3

En retirant les valeurs (mais en gardant les indices), on obtient la permutation (7401632859).

La plus petite valeur se trouve donc à la ligne 7, la deuxième plus petite à la ligne 4, etc.

Retrouver une valeur donnée se fait ainsi facilement par dichotomie.

Lorsqu'on stocke une table, il est très pratique de stocker et maintenir à jour le nombre total de lignes dont elle est constituée.

Dans une base de données relationnelle, il y a habituellement plusieurs tables reliées entre elles par des jeux de clés primaires, clés étrangères.

Comme expliqué précédemment, une clé primaire est une colonne ou un ensemble ordonné de colonnes ne pouvant prendre la ou les mêmes valeurs en deux lignes distinctes. (L'indice de ligne est un exemple fondamental de clé primaire.)

Supposons qu'une table soit constituée de plusieurs millions de lignes, mais que certaines de ses colonnes ne

puissent prendre qu'un nombre très réduit de valeurs différentes (par exemple, une base de données contenant des données de généalogie peut contenir les noms de personnes, pour chacune d'elles son pays de naissance, son continent de naissance, les pays et continents de naissance de sa mère et de son premier enfant s'il en a. Au lieu de renseigner toutes ces colonnes, il est considéré comme très économique de stocker dans un tel cas les pays dans une table séparée de la table principale et le continents dans une troisième table. La table principale contient alors à chaque ligne une valeur (une clé étrangère) donnant un identifiant de ligne (une valeur de clé primaire) de la table ``pays'' et la table ``pays'' contient, à chacune de ses lignes, une valeur (une clé étrangère) identifiant une des lignes de la table ``continent'' (clé primaire).

Voici un exemple (table clients ci-dessous) miniature illustrant ce qui précède :

(li)	Cn	Inc	Bircoun	BirCont	MoCoun	MoCont	EldCoun	EldCont
(0)	Dupont	817	France	Europe	Tunisia	Africa	England	Europe
(1)	Gracamoto	1080	Japan	Asia	Japan	Asia	USA	America
(2)	Smith	934	England	Europe	India	Asia	England	Europe
(3)	Helmut	980	Germany	Europe	Germany	Europe	Germany	Europe

(dans cet exemple, ``cn'' désigne le nom, ``inc'' le revenu, ``BirCoun'' le pays de naissance, ``BirCont'' le continent de naissance, ``MoCoun'' le pays de naissance de la mère, ``MoCont'' le continent de naissance de la mère, ``EldCoun'' le pays de naissance de l'aîné et ``EldCont'' le continent de naissance d'aîné.

Cette table peut être réécrite en plusieurs tables :

Continents:

(li)	Continent
(0)	Africa
(1)	America
(2)	Asia
(3)	Europe

Pays :

(li)	Country	Continent
(0)	France	3
(1)	Tunisia	0
(2)	England	3
(3)	Japan	2
(4)	USA	1
(5)	India	2
(6)	Germany	3

La table principale devient ainsi :

(li)	cn	Inc	Bircoun	MoCoun	EldCoun
(0)	Boyer	817	0	1	2
(1)	Gracamoto	1080	3	3	4
(2)	Smith	934	2	5	2
(3)	Helmut	980	6	6	6

L'ensemble des trois tables ainsi construites occupe certes moins de place que la table initiale.

Mais cela illustre aussi l'idée selon laquelle une base relationnelle peut être transformée en un ensemble de tables indépendantes les unes des autres.

Dans l'exemple précédent, on peut considérer la table ``Continents'' par elle-même, la table ``Pays'' avec la table ``continent'' développée dedans (c'est-à-dire la table ``pays'' dans laquelle les références à la table ``continents'' ont été remplacées par les lignes de la table elle-même) et la table ``Personnes'' avec les tables ``Pays'' et ``Continents'' développées dedans.

Les tables d'expansion sont alors :

Table d'expansion Continents :

(li)	Continent
(0)	Africa
(1)	America
(2)	Asia
(3)	Europe

La table d'expansion Pays devient :

(li)	Country	Continent
(0)	France	Europe
(1)	Tunisia	Africa
(2)	England	Europe
(3)	Japan	Asia
(4)	USA	America
(5)	India	Asia
(6)	Germany	Europe

Table d'expansion Clients :

(li)	Cn	Inc	Bircoun	BirCont	MoCoun	MoCont	EldCoun	EldCont
(0)	Boyer	817	France	Europe	Tunisia	Africa	England	Europe
(1)	Gracamoto	1080	Japan	Asia	Japan	Asia	USA	America
(2)	Smith	934	England	Europe	India	Asia	England	Europe
(3)	Helmut	980	Germany	Europe	Germany	Europe	Germany	Europe

Il peut bien évidemment arriver, comme dans cet exemple, qu'une table soit amenée à se développer plusieurs fois dans une autre. Cela signifie qu'une colonne de table développée dans une autre devra toujours être référencée comme appartenant à la table d'expansion, développée dans la table d'expansion via un jeu de clés primaires et clés étrangères qu'ils feront partie de l'identité de la colonne.

Nous définissons donc une table d'expansion comme une table dans laquelle toutes les tables qui pouvaient être développées en elle l'ont été, en autant d'exemplaires qu'il y a de jeux de clés primaires clés étrangères permettant de passer de la table d'expansion à la table développée.

Désormais, nous considérerons donc que la base de données relationnelle est formée de tables d'expansion indépendantes les unes des autres.

Pour chacune de ces tables d'expansion, on peut bâtir les index comme expliqué dans le cas d'une table seule.

Nous sommes maintenant en mesure d'interroger et de modifier notre base de données ainsi indexée.

Dans cette partie, nous expliquons comment les index créés peuvent être utilisés pour résoudre efficacement des requêtes SQL. Habituellement, une requête implique plusieurs tables et peut être séparée en deux parties distinctes : la clause ``where'' qui demande au gestionnaire de bases de

données de calculer des indices de lignes d'une table et une partie demandant au gestionnaire de bases de données de faire des calculs sur des données se trouvant aux lignes calculées.

La première partie peut contenir des jointures de tables (un lien entre une clé étrangère et la clé primaire correspondante), une comparaison entre une colonne et une constante (avec un connecteur arithmétique comme =, >=, >, <, <=, Between , like, in…) ou une comparaison entre deux colonnes (mêmes opérateurs arithmétiques ou encore un produit cartésien). Ces requêtes sont connectées entre elles, lorsqu'il y en a plusieurs, par des connecteurs logiques (and, or, not…).

La deuxième partie de la requête peut contenir des opérations arithmétiques comme des sommes, des moyennes, des produits, l'opérateur de dénombrement *, etc.

Comme expliqué précédemment, chacune des tables est considérée comme table d'expansion, ce qui signifie que les jointures de tables ne sont pas pertinentes pour une telle table.

Mais une requête comporte habituellement plusieurs tables. Comment choisir la table d'expansion dans laquelle résoudre la requête ?

Les tables impliquées dans la requête sont toutes développées dans un ensemble non vide, disons T.

Une seule de ces tables d'expansion n'est pas développée dans les autres. Cette table est la table dans laquelle nous devons résoudre la requête.

La clause ``where'' contient donc des clauses de jointures, reliées logiquement au reste de la requête par des connections logiques ``et''. Il suffit donc de tout simplement les effacer en remplaçant toute la clause ``et'' par l'autre terme. Cela signifie qu'on remplace ``(Jointure ET Reste)'' par ``Reste'' et ce pour chaque clause de jointure.

Voyons à présent comment gérer efficacement une clause ``where'' débarrassée de ses clauses de jointure.

Nous appelons ``requête atomique'' une portion indivisible de la clause where, c'est-à-dire une comparaison qui forme toute la requête ou qui est reliée au reste de la requête par des connecteurs logiques sans qu'elle en comporte elle-même. Si une table *t* comporte la colonne *c*, une requête atomique sera par exemple *t.c* = 3, *t.c* between ``HIGH'' and ``MEDIUM'', ou encore *t.c* like Word%.

Les prochains sous-paragraphes expliquent comment gérer les requêtes atomiques.

Ce cas est le plus simple à traiter est celui où il y a égalité entre une colonne et une valeur donnée. Il suffit de lire l'arbre à radicaux de la valeur recherchée pour la colonne de la requête.

La clause « Between » est un exemple fondamental de requête atomique. Toutes les autres requêtes atomiques peuvent se ramener à ce cas. C'est pour cette clause que les macro-mots ont été créés.

Reprenons l'exemple des dates données dans le paragraphe sur les macro-mots. Cette colonne a été gérée en enrichissant le vocabulaire des troncatures de ses mots de

longueur 4 et de longueur 6. Si on cherche les lignes dont la valeur est comprise entre [19931117, 19950225], il suffit de découper l'intervalle en : [19931117, 19950225] = [19931117, 19931130] U [199312, 199312] U [1994, 1994] U [199501, 199501]U [10050201, 19950225].

Le calcul est alors très simple : on lit l'arbre à radicaux de la valeur 19931117, qu'on unit (opération logique ``or'') avec celui de la valeur 19931118, … qu'on unit avec celui de la valeur 19931130 puis avec celui de la valeur (tronquée à 6 caractères) de 199312 puis avec celui (tronqué à 4 caractères) de 1994 puis avec celui (tronqué à 6 caractères) de 199501, puis avec celui de 19950201 puis avec celui de … 19950225.

Ainsi est-on amené à lire 42 arbres à radicaux au lieu des 466 qu'il aurait fallu lire sans les macro-mots.

Le traitement des ``or'' est expliqué plus bas.

On peut bien sûr traiter des intervalles semi-ouverts ou ouverts en excluant simplement les mots correspondants.

Chacune des requêtes atomiques « Supérieur ou égal, inférieur ou égal, supérieur, inférieur » est une clause between qui s'ignore. En effet, si on appelle m et M le minimum et maximum du thesaurus de la colonne concernée, alors

t.c > a	Signifie t.c est dans	]a,M]
t.c >= a	Signifie t.c est dans	[a,M]
t.c < a	Signifie t.c est dans	[m,a[
t.c <= a	Signifie t.c est dans	[m,a]

On peut ainsi traiter ces clauses comme une clause between.

La clause In est une façon de mixer des égalités par des ``or''. On peut donc les gérer très simplement.

Par exemple, *t.c* in (a,b,c) peut se réécrire en *t.c* = a or *t.c* = b or *t.c* = c. La gestion des clauses ``or'' est expliquée plus bas.

La clause ``like'' est un autre exemple de clause between. Par exemple, la clause *t.c* like Mot% se réécrit en effet en *t.c* between [Mot, Mou[.

Les requêtes atomiques peuvent être mixées à l'aide de connecteurs logiques : le ``And'', le ``Or'' et le ``Not''. Les trois prochaines sous-sections sont dédiées à ces opérateurs.

Nous souhaitons d'abord insister sur le fait qu'une requête atomique renvoie toujours en arbre à radicaux, ce qui sera aussi le cas de ces opérateurs logiques et pour finir de la clause where.

Le fait de faire un ``or'' (``ou'') de deux arbres à radicaux est en fait l'opération consistant à les réunir.

Ce calcul peut très aisément se faire par un parcours en largeur des deux arbres simultanément.

Il se fait récursivement par

```
union(t1, t2)
Debut
Arbre res;
Si (t1 = NULL) res = t2
Si (t2 = NULL) res = t1
```

```
        res->FilsGauche = Union(t1->FilsGauche, t2->FilsGauche)
        res->FilsDroit = Union(t1->FilsDroit, t2->FilsDroit)
        Renvoyer res
        Fin
```

La clause « and » se calcule presque comme la précédente (elle correspond à une intersection) :

```
        Intersection(t1, t2)
        Debut
        Arbre res;
        Si (t1 = NULL) res = NULL
        Si (t2 = NULL) res = NULL
        res->FilsGauche = Intersection(t1->FilsGauche, t2->FilsGauche)
        res->FilsDroit = Intersection(t1->FilsDroit, t2->FilsDroit)
        renvoyer res
        Fin
```

Cette clause demande néanmoins moins de temps de calcul en moyenne que la précédente. En effet, lors des deux parcours des arbres en parallèle, il suffit qu'un des deux noeuds explorés n'ait pas de fils gauche pour que l'exploration du fils gauche de l'autre noeud soit inutile.

Cela est particulièrement vrai quand les arbres ont été stockés sur disque dur dans des fichiers séparés.

La clause « Not » est une des plus difficiles à mettre en oeuvre parmi les requêtes atomiques. Elle peut néanmoins se traiter assez facilement.

L'indice maximal des lignes de chacune des tables est stocké et mis à jour. La clause Not peut alors se traiter comme suit (le but est de calculer l'arbre à radicaux Not T avec T un arbre à radicaux.

On définit ici un *n*-arbre à radicaux complet un arbre à radicaux contenant toutes les valeurs des entiers compris entre 0 et *n*-1.

Pour calculer un ``not'', il suffit alors de retirer récursivement, à l'aide d'un *x-or* les feuilles de *T* d'un *n* arbre à radicaux (où *n* désigne l'indice maximal des lignes de la table d'expansion à laquelle appartient *T*).

Lorsqu'on retire un noeud d'un arbre à radicaux, on le retire et on retire récursivement son père s'il n'a plus de fils.

Par exemple, la figure 3 montre le calcul de *Not T* lorsque la table d'expansion à laquelle *T* appartient à un indice maximum de lignes utilisé égal à 13.

L'arbre initial est présenté figure 3 et l'arbre transformé est présenté figure 4.

La comparaison entre deux colonnes est la plus complexe des requêtes atomiques. Cette requête se traite pratiquement comme un produit cartésien (voir la section suivante).

Soit *t* une table d'expansion et soient *t.c* et *t.d* deux de ses colonnes? Une comparaison entre ces deux colonnes est une opération au cours de laquelle on souhaite discriminer les lignes de *t* pour lesquelles *t.c* > *t.d* par exemple. Nous insistons sur le fait que cette comparaison se fait à indice

de ligne identique pour les deux colonnes, ce qui distingue cette comparaison du produit cartésien.

Comment pouvons-nous alors effectuer cette requête ?

Nommons T_c et T_d les thesaurus des deux colonnes *t.c* et *t.d*.

Nous recherchons les lignes telles qu'à ces lignes, *t.c* > *t.d*. Voici comment procéder. Pour tout mot *w* du thesaurus T_c, on calcule l'arbre à radicaux *r* de l'intervalle [*m_d, w'*] où *w'* désigne le plus grand mot de T_d inférieur strictement à *w*. Alors en calculant un ``et'' sur *r* et l'arbre à radicaux de *w*, on obtient un arbre t_w.

En effectuant la réunion de tous les arbres à radicaux r_w, on obtient l'arbre à radicaux recherché.

Il est bien évident que les arbres t_w ne doivent pas être calculés indépendamment les uns des autres. Puisque les mots *w* sont parcourus dans l'ordre croissant, il suffit d'unir t_w à l'arbre correspondant à l'ajout des mots compris entre *w* et le mot suivant dans T_c.

On peut aussi calculer t_{c-} t_d à l'aide des fichiers à plat et lire le résultat.

Les autres clauses semblables se résolvent de façon similaire (par exemple *t.c* >= t_d).

Cette sous-section et la suivante sont consacrées aux sous-requêtes.

En effet, il peut se trouver qu'une clause ``where'' contienne elle-même une autre clause ``where'', qui peut ou non être corrélée à la clause ``where'' principale.

Qu'est ce qu'une sous-requête corrélée ? Un exemple d'une telle requête est donné par la requête 17 du TPC. Cette requête est :

```
select
sum(l_extendedprice) / 7.0 as avg_yearly
from
 lineitem
 part
where
  p_partkey = l_partkey
  and p_brand = '[BRAND]'
  and p_container = '[CONTAINER]'
  and l_quantity < (
    select
     0.2 * avg(l_quantity)
    from
     lineitem
    where
     p_partkey = p_partkey
```

Dans cette requête, on doit réaliser le calcul d'une sous-requête tenant compte des conditions requises par la requête principale (puisque le p_partkey de la sous-requête appartient à la clause where principale).

Donc ce genre de requête peut être réécrite de façon à changer cette sous-requête en une sous-requête non corrélée. Il suffit pour cela de dupliquer les conditions requises par la clause where principale dans la clause where de la sous-requête corrélée. Dans notre exemple, cela donne :

```
select
   sum(l_extendedprice) / 7.0 as ag_yearly
```

```
from
  lineitem
  part
where
  p_partkey = l_partkey
  and p_brand = '[BRAND]'
  and p_container = '[CONTAINER]'
  and l_quantity < (
    select
      0.2 * avg(l_quantity)
    from
      lineitem
      partsupp
    where
      p_partkey = p_partkey
      and p_brand = '[BRAND]'
      and p_container = '[CONTAINER]'
  );
}
```

Finalement, une sous-requête corrélée peut se réécrire en une sous-requête non corrélée. C'est le sujet de la prochaine sous-section.

Une requête SQL contenant des sous-requêtes non corrélées peut se traiter en traitant d'abord les sous-requêtes récursivement et en remplaçant dans la requête la sous-requête par son résultat.

Désormais, nous sommes en mesure de traiter une clause ``where'' qui nous renvoie un arbre à radicaux représentant les indices de lignes de la table d'expansion correspondant à cette clause.

Supposons à présent que l'objet de la requête ait été de réaliser certains calculs sur quelques colonnes aux lignes trouvées. Par exemple on peut vouloir calculer la moyenne aux lignes trouvées des valeurs d'une certaine colonne.

Les valeurs de cette colonne sont stockées ``à plat'' dans leur ordre d'apparition. Il est donc très simple de relire les valeurs de cette colonne uniquement pour les lignes trouvées précédemment et effectuer sur ces valeurs les calculs demandés.

L'invention est décrite dans ce qui précède à titre d'exemple. Il est entendu que l'homme du métier est à même de réaliser différentes variantes de l'invention sans pour autant sortir du cadre du brevet.

REVENDICATIONS

1. Procédé d'organisation d'une base de données relationnelle destiné à être mis en œuvre sur une architecture informatique comprenant au moins un processeur et de la mémoire, caractérisé en ce qu'il comporte les étapes consistant à :

- élaborer une table d'expansion hiérarchique ;
- créer un thésaurus sur chacune des colonnes ;
- pour chacun des mots de chacun des thésaurus, créer l'arbre des radicaux de l'ensemble des indices de lignes sur lesquelles ledit mot apparaît ;
- pour chacune des clés primaires, stocker la suite de ces valeurs en mettant en œuvre une permutation sur l'ensemble de ces valeurs dans le but de retrouver une donnée.

2. Procédé d'organisation d'une base de données selon la revendication 1, caractérisé en ce qu'il comporte en outre une étape de découpage des tables de la base de données en un ensemble de sous-tables, chacune comportant un nombre donné de lignes à l'exception de la dernière sous-table.

3. Procédé d'organisation d'une base de données selon la revendication 1 ou 2, caractérisé en ce que la base de données met en œuvre le langage SQL (Structured Query Language).

4. Système de base de données organisé selon le procédé d'organisation de la revendication 1, 2 ou 3.

5. Procédé de requêtage d'une base de données organisée selon le procédé de la revendication 1, 2 ou 3, caractérisé en ce qu'il comporte

- une première étape de calcul d'une table d'expansion ;
- résoudre la clause « Where » de la requête en interrogeant les colonnes de ladite table d'expansion ;
- interroger les images non inversées des colonnes pour résoudre la clause « Select ».

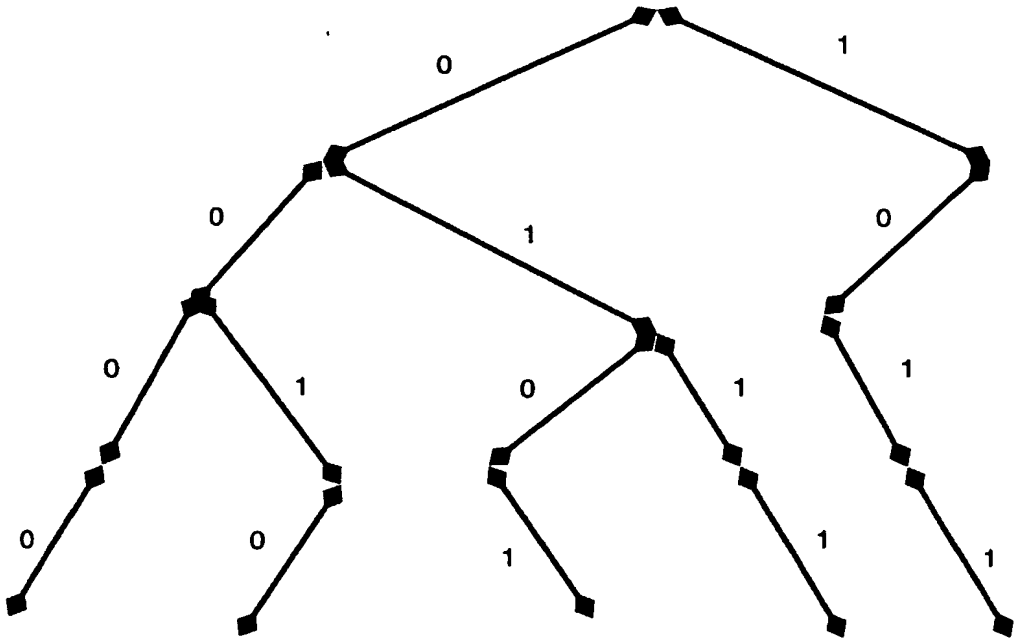
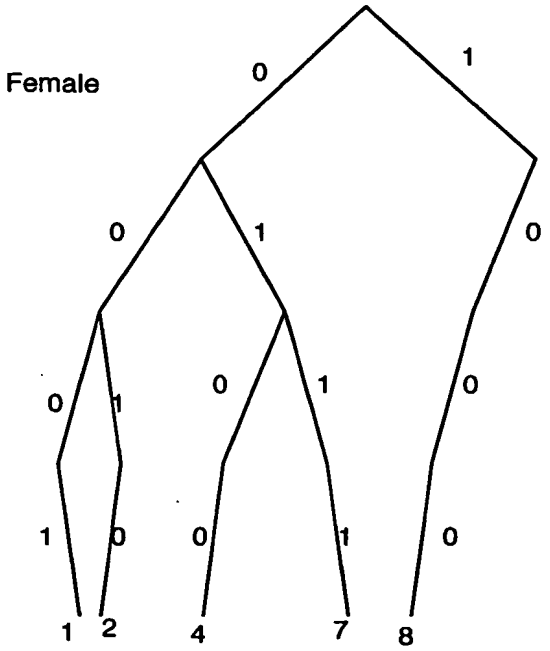


Figure 1


Female
0
1
0
1
0
0
1
0
1
0
1
0
1
0
1
2
4
7
8

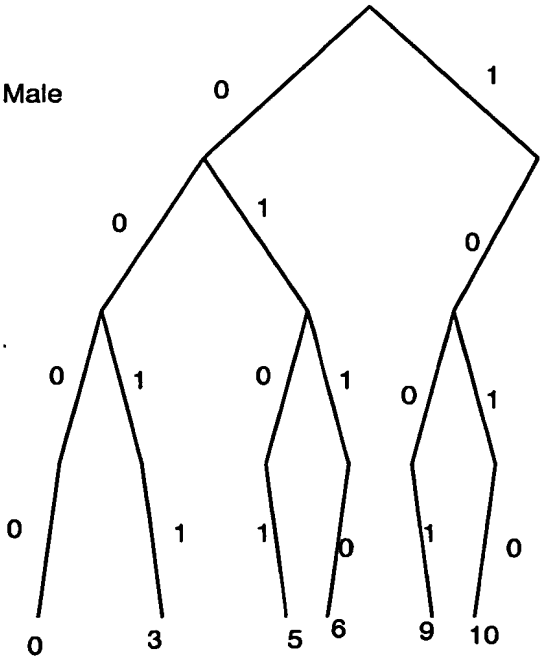

Male
0
1
0
1
0
0
1
0
1
0
1
0
1
0
1
0
1
0
3
5
6
9
10

Figure 2

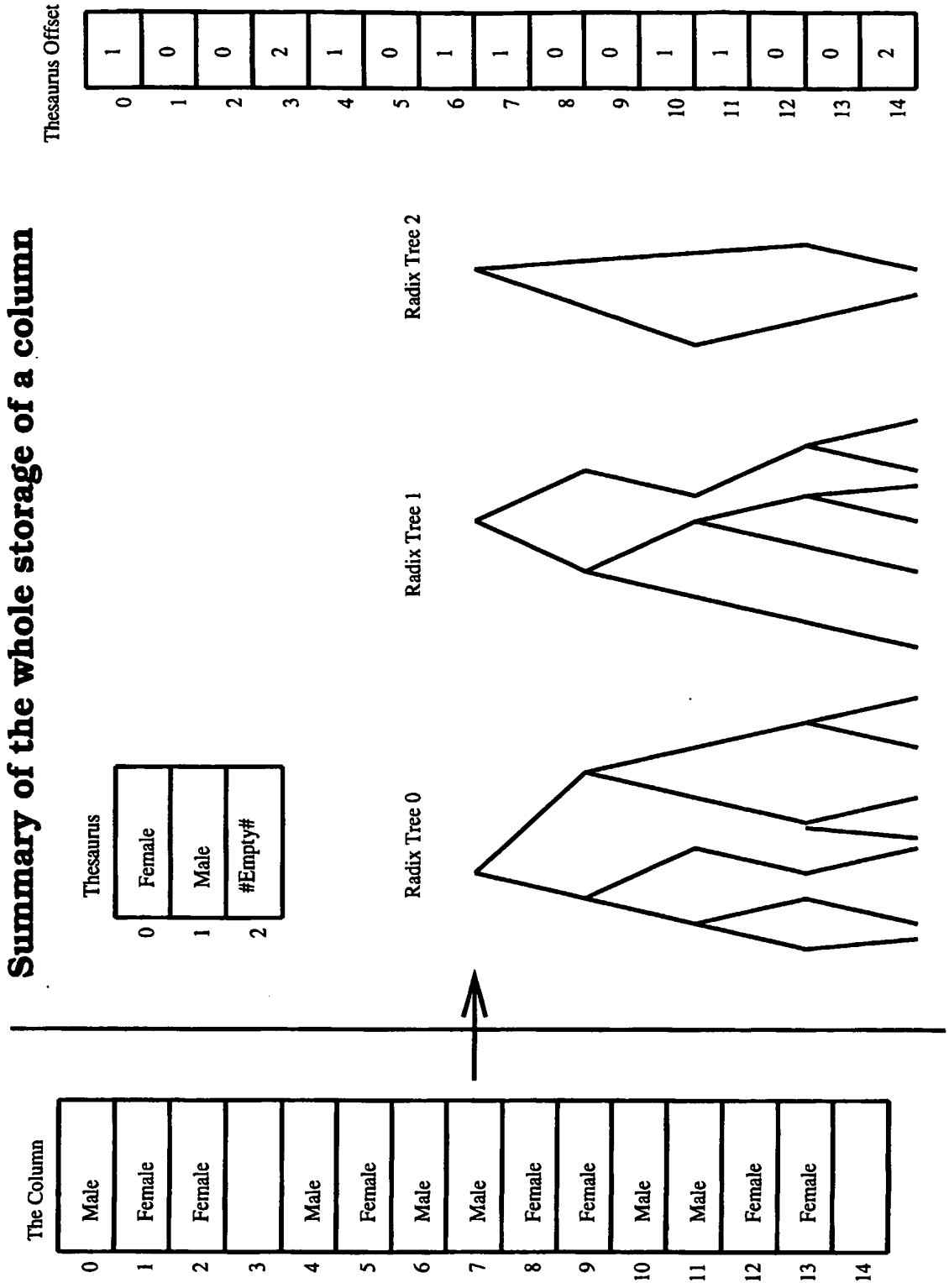

Summary of the whole storage of a column
The Column
0 Male
1 Female
2 Female
3
4 Male
5 Female
6 Male
7 Male
8 Female
9 Female
10 Male
11 Male
12 Female
13 Female
14
Thesaurus
0 Female
1 Male
2 #Empty#
Radix Tree 0
Radix Tree 1
Radix Tree 2
Thesaurus Offset
0 1
1 0
2 0
3 2
4 1
5 0
6 1
7 1
8 0
9 0
10 1
11 1
12 0
13 0
14 2

Figure 3

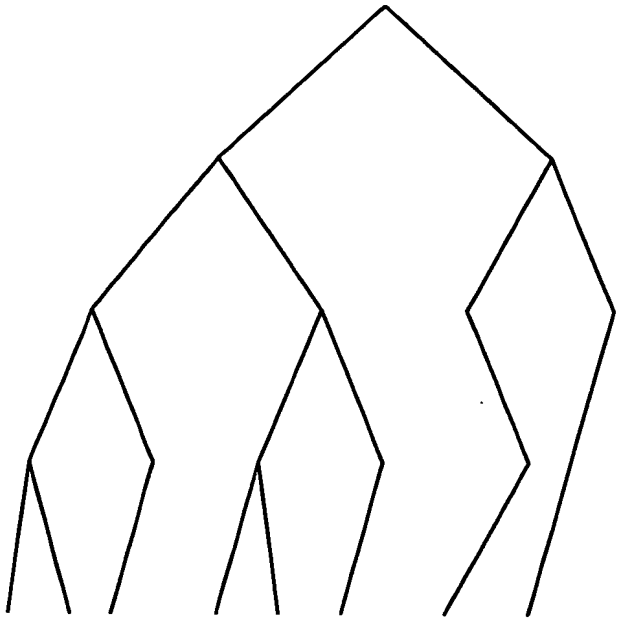


Figure 4

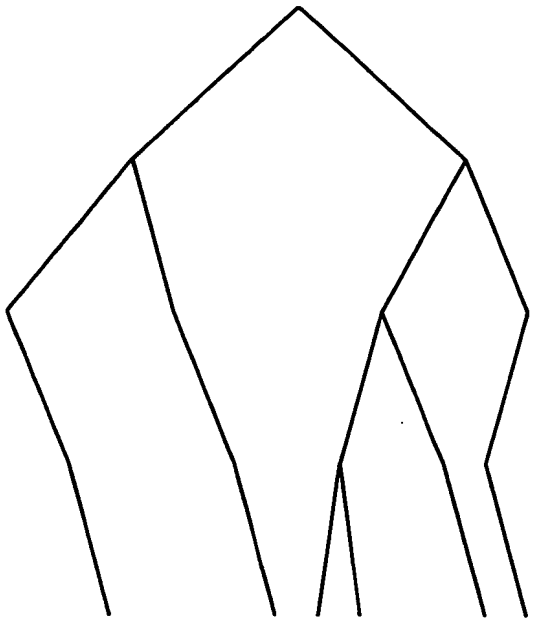


Figure 5

INTERNATIONAL SEARCH REPORT

International Application No
P[redacted]/FR2004/002371

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 1 217 540 A (LAFAYETTE SOFTWARE INC) 26 June 2002 (2002-06-26) paragraph '0005! - paragraph '0008! figures 1-16	1-5
X	US 6 266 662 B1 (OZBUTUN CETIN ET AL) 24 July 2001 (2001-07-24) column 5, line 30 - column 7, line 59 figures 3a,3b	1-5
A	D.E. KNUTH: "The Art of Computer Programming, Volume 3" 1973, ADDISON-WESLEY , READING, MASSACHUSETTS, USA , XP002321286 page 11 - page 33	1-5

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 March 2005	31/03/2005

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Eichenauer, L

INTERNATIONAL SEARCH REPORT

nformation on patent family members

International Application No
PCT/FR2004/002371

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 1217540 A	26-06-2002	EP 1217540 A1	26-06-2002
		AU 3203502 A	11-06-2002
		CA 2430333 A1	06-06-2002
		EP 1423799 A2	02-06-2004
		WO 0244943 A2	06-06-2002
		JP 2004534981 T	18-11-2004
		US 6711563 B1	23-03-2004
US 6266662 B1	24-07-2001	US 6195656 B1	27-02-2001
		US 6067540 A	23-05-2000

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
CIB 7 G06F17/30

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
CIB 7 G06F

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si réalisable, termes de recherche utilisés)
EPO-Internal

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
X	EP 1 217 540 A (LAFAYETTE SOFTWARE INC) 26 juin 2002 (2002-06-26) alinéa '0005! - alinéa '0008! figures 1-16	1-5
X	US 6 266 662 B1 (OZBUTUN CETIN ET AL) 24 juillet 2001 (2001-07-24) colonne 5, ligne 30 - colonne 7, ligne 59 figures 3a,3b	1-5
A	D.E. KNUTH: "The Art of Computer Programming, Volume 3" 1973, ADDISON-WESLEY , READING, MASSACHUSETTS, USA , XP002321286 page 11 - page 33	1-5

☐ Voir la suite du cadre C pour la fin de la liste des documents

☒ Les documents de familles de brevets sont indiqués en annexe

° Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'inven tion revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'inven tion revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

Date à laquelle la recherche internationale a été effectivement achevée	Date d'expédition du présent rapport de recherche internationale
15 mars 2005	31/03/2005

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Eichenauer, L

Document brevet cité au rapport de recherche		Date de publication	Membre(s) de la famille de brevet(s)		Date de publication
EP 1217540	A	26-06-2002	EP	1217540 A1	26-06-2002
			AU	3203502 A	11-06-2002
			CA	2430333 A1	06-06-2002
			EP	1423799 A2	02-06-2004
			WO	0244943 A2	06-06-2002
			JP	2004534981 T	18-11-2004
			US	6711563 B1	23-03-2004
US 6266662	B1	24-07-2001	US	6195656 B1	27-02-2001
			US	6067540 A	23-05-2000

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☑ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.